



AcroTeX.Net

## AcroTeX PDF Blog

### Processing Acrobat Forms using JavaScript

#### Internal Processing of a Field

#### Part 4: The Validate Tab

D. P. Story

**Warning:** In Section 18.1 we discuss the (undocumented) folder-level JavaScript that ship with Acrobat/AR. The ones discussed have all been around since version 5, and the extended versions since version 6. While there is always a desire on the part of the Acrobat engineering team to maintain compatibility with previous versions, changes can occur. By using these functions directly you run the risk—slight as it might be—of incompatibilities with future versions of Acrobat. ⚠

## Table of Contents

<b>18 Validate tab for the Text Field and Editable Combo Box</b>	<b>3</b>
18.1 Built-in Validate Script . . . . .	4
18.2 Custom Validate Script . . . . .	5

## 18. Validate tab for the Text Field and Editable Combo Box

In [AcroTeX Blog #20](#), we investigated the Action tab; in that article, the events of Mouse Enter, Mouse Up, Mouse Down, and Mouse Exit are triggered by direct user interaction with the document. In [AcroTeX Blog #21](#), we explored the chain of events—Keystroke, Validate, Calculate, and Format—that occur when a user enters data into a text field or editable combo box, and surveyed the important properties of the event object for each of these events.

Each of the events (Keystroke, Validate, Calculate, and Format) can have a script attached to it:

- Keystroke and Format event: Custom scripts are entered through the Format tab; in addition to custom script, there are several built-in scripts that ship with Acrobat/AD, these are available through the Format tab user interface.
- Validate event: A custom script can be entered through the Validate tab; there is also a built-in script for validating common requirements, this script is available through the Validate tab.
- Calculate tab: a custom script can be entered through the Calculate tab; also included in this tab are some built-in scripts for making common calculations, these scripts are available through the user interface of the Calculate tab.

In this article, we study the scripting associated with the Validate tab. We shall discuss custom Validate scripts, as well as survey the built-in scripts that ship with Acrobat/AD and are available through the user interface.

**What is Validation?** Text input may have restrictions on them, whereas, these any restrictions may be checked during the Keystroke event, when `event.willCommit` is true, Acrobat has compartmentalized the event chain, allowing the script writer to write general script for validating user input. This allows for the creation of “libraries” of keyboard input script, and “libraries” of validation scripts—much like the built-in script that are available through the user input. Currently, Acrobat only has one built-in Validation script, this is used to force a field that takes numerical input to be within a specific range of values. We look at the JavaScript counter-part to this in [Section 18.1](#), page 4. See [Figure 1](#) on page 4 for a screen shot of the Validate tab.

Though the Validate script is designed for “validation” of user input, it can, in fact, be used for whatever purpose you can conceive of.

**Acknowledgements:** I extracted some of the information found in Section 18.1 by snooping through a PDF file using the [PDF CanOpener](#), by [WindJack Solutions, Inc.](#)<sup>1, 2</sup>

<sup>1</sup><http://www.windjack.com>

<sup>2</sup>I downloaded a fully functional version of the product, an Acrobat plug-in, for a 10 day trial period. I found

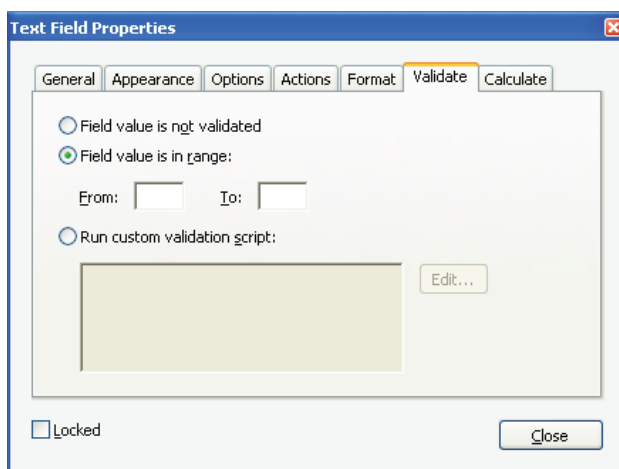


Figure 1: The Validate Tab

Section 2.3.6.1 of *The L<sup>A</sup>T<sub>E</sub>X Web Companion, Integrating T<sub>E</sub>X, HTML, and XML*, by Michel Goossens, Sebastian Rahtz, *et al*, was also used as a resource for Sections 18.1. This book is one of the few that documents (at least partially) the folder level JavaScript functions that are used by the built-in scripts of the Format, Validate, and Calculate tabs.

### 18.1. Built-in Validate Script

Let's look at the built-in Validate script first, because one of the examples presented in [Section 18.2](#), page 5, uses the built-in script.

The one built-in Validate script verifies whether the value of a text field (or editable combo box), which takes numerical input, is in a specified range, see [Figure 1](#) to view the Validate tab of a Text Field Properties dialog box. The user interface is implemented through the folder JavaScript function, `AFRange_Validate`. The parameters of this function are shown below:

```
AFRange_Validate(bGreaterThan, nGreaterThan, bLessThan, nLessThan)
```

Let me illustrate the argument by example:

- To test if  $a \leq \text{event.value} \leq b$ , use the arguments:

```
AFRange_Validate(true, a, true, b)
```

- To test if  $a \leq \text{event.value}$ , use the arguments:

the [PDF CanOpener](#) to be intuitive and very easy to use. It is indeed an excellent tool for examining (and editing) the structure of a PDF file.

```
AFRange_Validate(true, a, false, 0)
```

The fourth parameter can be any number, it is not used, but it must be present.

- To test if `event.value ≤ b`, use the arguments:

```
AFRange_Validate(false, 0, true, b)
```

The first parameter can be any number, it is not used, but it must be present.

### Example 18.1. Requiring numerical input to be in a certain range.

```
Validate script: AFRange_Validate(true, 1.5, true, 5.5)
```

```
Validate script: AFRange_Validate(true, 1.5, false, 0)
```

```
Validate script: AFRange_Validate(false, 0, true, 5.5)
```

The Keystroke and Format scripts used the two built-in functions `AFNumber_Keystroke` and `AFNumber_Format`, as explained in [AcroTeX Blog #22](#).



## 18.2. Custom Validate Script

The Validate event comes just after the Keystroke event,<sup>3</sup> the value entered by the user in the Keystroke event is available in the Validate script as `event.value`. Within the Validate script, the user input can be invalidated by setting `event.rc` to `false`, in this case, the entire input of the user is rejected, and the last cached (and validated) value of the field is used.

Let's create a custom Validate script that is a variation on what is available through the user interface. In the next example, we require the user to enter a date, but the date is required to fall into a range of dates.

**Example 18.2. Date Range Validation.** The following three fields are similar to the three field found in [Example 18.1](#) on page 5, but we validate the range of the date entered.

The Validate script for each of these fields is, respectively,

```
APDFBDate_Validate("m/d/yy", true, "1/1/05", true, "12/17/09")
```

```
APDFBDate_Validate("m/d/yy", true, "1/1/05", false, "12/17/09")
```

```
APDFBDate_Validate("m/d/yy", false, "1/1/05", true, "12/17/09")
```

<sup>3</sup>Provided `event.rc=true` in the Keystroke event.

The function `APDFBDate_Validate` is a custom validate function that appears at the document level. The verbatim listing is given below:

```

1  function APDFBDate_Validate(cFormat,bGreaterThan,cGreaterThan,
2     bLessThan, cLessThan) {
3     if (event.value == "") return;
4     var d_l, d_u, d;
5     var cError = "";
6     d=AFParseDateEx(event.value, cFormat);
7     if (bGreaterThan) {
8         d_u=AFParseDateEx(cGreaterThan, cFormat);
9         if ( d_u == null ) {
10            console.show();
11            console.println("Author: Bad upper date bound ["
12                +cGreaterThan+"]");
13            event.rc=false; return;
14        };
15    }
16    if (bLessThan) {
17        d_u=AFParseDateEx(cGreaterThan, cFormat);
18        if ( d_u == null ) {
19            console.show();
20            console.println("Author: Bad upper date bound ["
21                +cGreaterThan+"]");
22            event.rc=false; return;
23        };
24    }
25    if (bGreaterThan && bLessThan) {
26        d_l=AFParseDateEx(cLessThan, cFormat);
27        if (d.valueOf() < d_u.valueOf() || d.valueOf() > d_l.valueOf() ) {
28            cError = util.printf(oMsgDate.GT_AND_LT,
29                cGreaterThan,cLessThan);
30        }
31    } else if (bGreaterThan) {
32        if (d.valueOf() < d_u.valueOf()) {
33            cError = util.printf(oMsgDate.GREATER_THAN,cGreaterThan);
34        }
35    } else if (bLessThan) {
36        d_l=AFParseDateEx(cLessThan, cFormat);
37        if ( d.valueOf() > d_l.valueOf() ) {
38            cError = util.printf(oMsgDate.LESS_THAN,cLessThan);
39        }
40    }
41    if (cError != "") {
42        app.alert(cError, 0); event.rc = false;
43    }
44 }

```

**Code Comments:** This function follows code lines of `AFRange_Validate`, with changes made appropriate to comparing dates rather than numbers. The error messages are given at the document level, see the section of the document JavaScript named **Date Validate Example**. We use an undocumented built-in function `AFParseDateEx`, its first parameter is the date string to be parsed, the second parameter is the date format `cFormat` to use. The function returns `null` if the string is not a date, and returns a date object if it is. If `d` is a date object, `d.valueOf()` is a number representing the number of milliseconds since the base time value. □

**Example 18.3. Clearing a Calculation Field on Reset.** On thing that has bothered me about calculation fields is that they don't reset to an empty value. See the set of calculation to the left. By using a Validate script, you can force the calculation field to be empty on reset, see the same set of fields to the right.

Enter a number in the two top rows, the third row is the sum of the two rows above it. Reset after entering your numbers, and observe the values of the calculate fields.

The code for the Validate script for the last field in the right column is

```
clearCalcFieldonReset("txtNum1-wReset","txtNum2-wReset");
```

The function `clearCalcFieldonReset` is defined in the document JavaScript section of the document, it is listed under the name **Clear Calculate Field on Reset**. The verbatim listing follows:

```
1 function clearCalcFieldonReset() {
2     for ( var bReset=true, i=0; i<arguments.length; i++){
3         var f=this.getField(arguments[i]);
4         if (f!=null && f.value!="") {bReset=false; break}
5     }
6     if (bReset) event.value="";
7 }
```

The parameters for `clearCalcFieldonReset` is a list of the field names that are used to calculate the value of the field, in this case, they are `"txtNum1-wReset"` and `"txtNum2-wReset"`. The function gets the field names from the function arguments, and loops through to see if any of these fields is non-empty; if all are empty, we set `event.value=""`. Here, `event.value` is the value of the field executing the Validate script. □

That was an example of using the Validate script for something other than what it was intended, now here's another.

**Example 18.4. Riddle me this:** Who has three legs in the morning, two legs in the afternoon, and four legs in the evening?

The Validate script contains the code for determining if your response is correct. Please, take a look!

Well, that's pretty much it for now, I simply must get back to my retirement. ☹